Cache Contention and Application Performance Prediction for Multi-Core Systems

Chi Xu*, Xi Chen, Robert P. Dick, Zhuoqing Morley Mao

*University of Minnesota, University of Michigan

IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS), March 2010
Motivation

Multiprocessor architectures (CMP) with shared last-level caches

- Inter-process communication
- Heterogeneous cache allocation
- Contention
Motivation

Multiprocessor architectures (CMP) with shared last-level caches
- Inter-process communication
- Heterogeneous cache allocation
- Contention

⇒ Performance implications of core assignment
Goal

- Model cache contention
- Easy and automatic
- No modifications to existing hardware or operating system
- No exhaustive offline simulation
- Complementary to existing work
Analytical Model — System

- N-core processor
- On-chip last-level L2 Cache
  - Set-associative ($ways = lines$ per set)
  - LRU replacement policy
  - Shared among cores
  - No Prefetching
- Applications in steady state
Analytical Model—Applications I

Effective Cache Size

\[
\sum_{i=1}^{N} S_i = A
\]

- \(N\): Total number of processes
- \(S_i\): Effective cache size of process \(i\) (ways occupied by \(i\))
- \(A\): Associativity of cache
Analytical Model—Applications I

**Effective Cache Size**

\[
\sum_{i=1}^{N} S_i = A
\]

- \(N\): Total number of processes
- \(S_i\): Effective cache size of process \(i\) (ways occupied by \(i\))
- \(A\): Associativity of cache

**Reuse Distance**

- The first bar in the histogram, i.e., \(1\) represents the number of distinct cache lines used 12 lines, which can be denoted as \(\sum_{i=1}^{\infty} \text{hist}_i(1)\).
- The last bar, i.e., \(13+\), gives the probability that the data in the line is never accessed again. Note that \(\text{hist}_i(13+)\) will be stable. We define this as the normalized frequencies of the associated reuse distances.
- Note that \(\int_{0}^{\infty} \text{hist}_i(x) dx\) gives the probability that the data in the line is never accessed again, while \(\int_{1}^{\infty} \text{hist}_i(x) dx\) gives the probability that the data in the line is not accessed for the next cache access does not exist in the most-recently-used line. A most-recently-used line will be accessed again.
- The cache contention prediction problem can be formulated as follows: given \(\text{hist}_i(x)\), predict the probability of cache miss for process \(i\) when \(x\) cache lines are allocated to \(i\) again.

The equation is:

\[
\int_{0}^{\infty} \text{hist}_i(x) dx = 1
\]

We make the following assumptions.

1. For each process, we assume that data accesses are uniformly distributed across all cache sets.
2. We assume no hardware prefetching. Hardware prefetching may be fetching cache lines based on access patterns. The analytical model might therefore be inaccurate for systems using hardware prefetching with an effective cache size of \(S\) (ways occupied by \(i\)).
### Effective Cache Size

\[ \sum_{i=1}^{N} S_i = A \]

- \( N \): Total number of processes
- \( S_i \): Effective cache size of process \( i \) (ways occupied by \( i \))
- \( A \): Associativity of cache

### Reuse Distance

\[ MPA_i(S_i) = \int_{S_i}^{\infty} hist_i(x) \, dx \]

- **MPA**: Probability of cache miss for process \( i \)
- **hist**: Linear interpolation of reuse distance histogram
### Cache Accesses

\[
\text{APS} = \frac{\text{API}}{\alpha \cdot \text{MPA} + \beta}
\]

- **APS**: Accesses per second
- **API**: Accesses per instruction (fixed for each application)
- **SPI**: Seconds per instruction

**Effective Cache Size**

\[
G_i(n) = n \sum_{s=1}^{\infty} (P_{s,n} \cdot s)
\]

\[
\text{G}_i(n) \quad \text{Effective cache size of process } i \text{ after } n \text{ accesses}
\]

\[
P_{s,n} \quad \text{Probability of having } s \text{ cache lines after } n \text{ consecutive accesses}
\]

\[
\text{steady} \quad \Downarrow
\]

\[
n = G - 1
\]

\[
G(n) \quad \text{steady state}
\]
### Cache Accesses

<table>
<thead>
<tr>
<th>APS</th>
<th>API</th>
<th>SPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>APS accesses per second</td>
<td>API accesses per instruction (fixed for each application)</td>
<td>SPI seconds per instruction</td>
</tr>
<tr>
<td>APS = ( \frac{API}{SPI} )</td>
<td>SPI = ( \alpha \cdot MPA + \beta )</td>
<td></td>
</tr>
</tbody>
</table>

- **APS** accesses per second
- **API** accesses per instruction (fixed for each application)
- **SPI** seconds per instruction
- \( \alpha \) off-chip latency (memory, disk)
- \( \beta \) on-chip latency (computation)
### Cache Accesses

<table>
<thead>
<tr>
<th>APS</th>
<th>Accesses per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Accesses per instruction</td>
</tr>
<tr>
<td>SPI</td>
<td>Seconds per instruction</td>
</tr>
</tbody>
</table>

**APS** = \( \frac{\text{API}}{\text{SPI}} \)

SPI = \( \alpha \cdot \text{MPA} + \beta \)

- \( \alpha \): Off-chip latency (memory, disk)
- \( \beta \): On-chip latency (computation)

### Effective Cache Size

\[
G_i(n) = \sum_{s=1}^{n} (P_{s,n} \cdot s)
\]

- \( G_i(n) \): Effective cache size of process \( i \) after \( n \) accesses
- \( P_{s,n} \): Probability of having \( s \) cache lines after \( n \) consecutive accesses
### Analytical Model—Applications II

#### Cache Accesses

<table>
<thead>
<tr>
<th>APS Accesses per second</th>
<th>APS = ( \frac{\text{API}}{\text{SPI}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>API Accesses per instruction</td>
<td>( \text{API} ) (fixed for each application)</td>
</tr>
<tr>
<td>Seconds per instruction</td>
<td>( \text{SPI} )</td>
</tr>
<tr>
<td>Off-chip latency (memory, disk)</td>
<td>( \alpha )</td>
</tr>
<tr>
<td>On-chip latency (computation)</td>
<td>( \beta )</td>
</tr>
</tbody>
</table>

\[ \text{SPI} = \alpha \cdot \text{MPA} + \beta \]

#### Effective Cache Size

\[ G_i(n) = \sum_{s=1}^{n} (P_{s,n} \cdot s) \]

- \( G_i(n) \): Effective cache size of process \( i \) after \( n \) accesses
- \( P_{s,n} \): Probability of having \( s \) cache lines after \( n \) consecutive accesses

\[ n = G_i^{-1}(S_i) \]
At time $t$ there is a duration $T$ such that data accessed... 
- before $t - T$ are evicted from cache
- during $[t - T, t]$ are present in cache
At time $t$ there is a duration $T$ such that data accessed...

- before $t - T$ are evicted from cache
- during $[t - T, t]$ are present in cache

Assuming all processes are in steady state:

$$S_i = G_i(APS_i \cdot T)$$
At time $t$ there is a duration $T$ such that data accessed...

- before $t - T$ are evicted from cache
- during $[t - T, t]$ are present in cache

Assuming all processes are in steady state:

$$S_i = G_i(APS_i \cdot T)$$

$\downarrow$

$$APS_i = G_i^{-1}(S_i) / T$$
At time $t$ there is a duration $T$ such that data accessed...
- before $t - T$ are evicted from cache
- during $[t - T, t]$ are present in cache

Assuming all processes are in steady state:

$$S_i = G_i(APS_i \cdot T)$$

$$APS_i = G_i^{-1}(S_i)/T$$

$$APS_i = \frac{G_i^{-1}(S_i)}{T} = \frac{\text{API}_i}{\alpha_i \cdot \text{MPA}_i(S_i) + \beta_i}$$

Reminder

$$APS = \frac{\text{API}}{\text{SPI}}$$

$$\text{SPI} = \alpha \cdot \text{MPA} + \beta$$
At time $t$ there is a duration $T$ such that data accessed...
- before $t - T$ are evicted from cache
- during $[t - T, t]$ are present in cache

Assuming all processes are in steady state:

$$S_i = G_i(\text{APS}_i \cdot T)$$

↓

$$\text{APS}_i = G_i^{-1}(S_i) / T$$

↓

$$\text{APS}_i = \frac{G_i^{-1}(S_i)}{T} = \frac{\text{API}_i}{\alpha_i \cdot \text{MPA}_i(S_i) + \beta_i}$$

↓

$$T = \frac{G_i^{-1}(S_i) \cdot \alpha_i \cdot \text{MPA}_i(S_i) + \beta_i}{\text{API}_i}$$

**Reminder**

$$\text{APS} = \frac{\text{API}}{\text{SPI}}$$

$$\text{SPI} = \alpha \cdot \text{MPA} + \beta$$
At time $t$ there is a duration $T$ such that data accessed...
- before $t - T$ are evicted from cache
- during $[t - T, t]$ are present in cache

Assuming all processes are in steady state:

\[ S_i = G_i(APS_i \cdot T) \]

\[ \downarrow \]

\[ APS_i = G_i^{-1}(S_i) / T \]

\[ \downarrow \]

\[ APS_i = \frac{G_i^{-1}(S_i)}{T} = \frac{API_i}{\alpha_i \cdot MPA_i(S_i) + \beta_i} \]

\[ \downarrow \]

\[ T = \frac{G_i^{-1}(S_i) \cdot \alpha_i \cdot MPA_i(S_i) + \beta_i}{API_i} \]

\[ \downarrow \]

\[ \forall j = 1^N : \frac{G_1^{-1}(S_1)}{G_j^{-1}(S_j)} = \frac{API_1 \cdot (\alpha_j \cdot MPA_j(S_j) + \beta_j)}{API_i \cdot (\alpha_1 \cdot MPA_1(S_1) + \beta_1)} \quad \text{and} \quad \sum_{i=1}^{N} S_i - A = 0 \]
Automated Profiling

- Two processes running on separate cores sharing A-way last-level cache
- One process uses $l$ ways $\Rightarrow$ other process uses $A - l$ ways
- \textit{stressmark}: synthetic application with configurable cache occupation
- Gather information on API, MPA and SPI via hardware performance counters
- Derive reuse distance histogram, effective cache size $(S)$, $\alpha$ and $\beta$
  $\Rightarrow$ application-dependent feature vector
Evaluation

- Intel Core 2 Duo-P8600 (2 core @ 2.4GHz, 3 MB 12-way associative L2 cache)
- MacOS X 10.5
- Profiling via Shark at a period of 2 ms
- Subset of SPEC CPU2000: 5 CPU-intensive + 5 memory-intensive
- Each application run 12 times for 10 s to determine characteristics
- Examine all 55 pairwise combinations
**Application profiles**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>art</th>
<th>mcf</th>
<th>bzip2</th>
<th>swim</th>
<th>equake</th>
<th>mesa</th>
<th>vpr</th>
<th>ammp</th>
<th>mgrid</th>
<th>applu</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>0.0225</td>
<td>0.0733</td>
<td>0.0044</td>
<td>0.0116</td>
<td>0.0074</td>
<td>0.0013</td>
<td>0.0102</td>
<td>0.0092</td>
<td>0.0018</td>
<td>0.0018</td>
</tr>
<tr>
<td>$\alpha$ ($\times 10^{-9}$)</td>
<td>446</td>
<td>134</td>
<td>99.9</td>
<td>-99.6</td>
<td>60.5</td>
<td>30.7</td>
<td>306</td>
<td>243</td>
<td>0.609</td>
<td>3.12</td>
</tr>
<tr>
<td>$\beta$ ($\times 10^{-7}$)</td>
<td>1.34</td>
<td>5.86</td>
<td>15.0</td>
<td>1.97</td>
<td>2.28</td>
<td>1.55</td>
<td>1.65</td>
<td>1.83</td>
<td>1.28</td>
<td>1.15</td>
</tr>
</tbody>
</table>

**Table II: API, $\alpha$, and $\beta$ for different benchmarks**

- **Effective Cache Size**
  - art
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12
  - mcf
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12
  - vpr
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12
  - mesa
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12
  - mgrid
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12
  - swim
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12
  - ammp
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12
  - applu
    - Miss Rate
      - 0
      - 2
      - 4
      - 6
      - 8
      - 10
      - 12

Figure 3. Profiled cache miss rate corresponding to effective cache size.

- **Equation used to determine $S$ (MB), assumes that**
  - Miss Rate
    - Values range from 0 to 1
    - X-axis represents effective cache size
    - Y-axis represents miss rate

- **Analysis of Results:**
  - We examined all 55 possible pairwise combinations of benchmarks.
  - In addition to the proposed technique, we considered three models:
    - Miss Rate
      - Based (MB)
      - Based (AB)
      - Misses Based (AB)
  - The first model, known as Miss Rate Based (MB), assumes the effective cache size of a process is available in real systems.
  - The second model, known as Misses Based (AB), assumes the effective cache size of a process is unknown.
  - Accuracy depends on the concurrently running processes. In contrast, our technique eliminates the dependency on such conditions.
  - The average prediction error for CAMP is $1.57\%$, compared to $21\%$ for AB and $33\%$ for MB.
  - The percentage of test cases with a prediction error larger than $5\%$ among all 10 test cases for CAMP is $10\%$, while $21\%$ for AB and $33\%$ for MB.

- **Columns:**
  - Columns 1-4 list the benchmarks.
  - Columns 5-8 list the percentage of test cases with a cache miss estimation error larger than $5\%$.
  - Columns 9-12 list the percentage of test cases with a relative performance estimation error larger than $5\%$.
  - Columns 13-16 indicate the average relative estimation error in performance.

- **Graphs:**
  - Each graph represents the profiled cache miss rate for a specific benchmark against effective cache size.
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CAMP</th>
<th></th>
<th></th>
<th></th>
<th>AB</th>
<th></th>
<th></th>
<th></th>
<th>MB</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPA</td>
<td>SPI</td>
<td>MPA</td>
<td>SPI</td>
<td>MPA</td>
<td>SPI</td>
<td>MPA</td>
<td>SPI</td>
<td>MPA</td>
<td>SPI</td>
<td>MPA</td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>&gt;5%</td>
<td>Error</td>
<td>&gt;5%</td>
<td>Error</td>
<td>&gt;5%</td>
<td>Error</td>
<td>&gt;5%</td>
<td>Error</td>
<td>&gt;5%</td>
<td>Error</td>
</tr>
<tr>
<td>art</td>
<td>1.61</td>
<td>0</td>
<td>3.68</td>
<td>40</td>
<td>4.60</td>
<td>50</td>
<td>10.26</td>
<td>80</td>
<td>5.88</td>
<td>70</td>
<td>18.09</td>
</tr>
<tr>
<td>vpr</td>
<td>0.88</td>
<td>0</td>
<td>1.48</td>
<td>0</td>
<td>4.70</td>
<td>40</td>
<td>7.67</td>
<td>60</td>
<td>5.89</td>
<td>30</td>
<td>9.24</td>
</tr>
<tr>
<td>mcf</td>
<td>2.10</td>
<td>10</td>
<td>3.70</td>
<td>20</td>
<td>2.82</td>
<td>10</td>
<td>3.97</td>
<td>40</td>
<td>6.79</td>
<td>40</td>
<td>7.72</td>
</tr>
<tr>
<td>ammp</td>
<td>2.82</td>
<td>20</td>
<td>3.04</td>
<td>20</td>
<td>4.03</td>
<td>30</td>
<td>4.16</td>
<td>30</td>
<td>5.89</td>
<td>60</td>
<td>6.78</td>
</tr>
<tr>
<td>bzip2</td>
<td>1.86</td>
<td>10</td>
<td>1.17</td>
<td>0</td>
<td>3.17</td>
<td>20</td>
<td>1.89</td>
<td>0</td>
<td>6.09</td>
<td>60</td>
<td>3.63</td>
</tr>
<tr>
<td>mesa</td>
<td>4.23</td>
<td>50</td>
<td>0.83</td>
<td>0</td>
<td>4.90</td>
<td>30</td>
<td>0.94</td>
<td>0</td>
<td>7.77</td>
<td>50</td>
<td>1.55</td>
</tr>
<tr>
<td>swim</td>
<td>0.28</td>
<td>0</td>
<td>0.86</td>
<td>0</td>
<td>0.23</td>
<td>0</td>
<td>0.81</td>
<td>0</td>
<td>0.27</td>
<td>0</td>
<td>0.78</td>
</tr>
<tr>
<td>equake</td>
<td>0.70</td>
<td>0</td>
<td>0.38</td>
<td>0</td>
<td>0.92</td>
<td>0</td>
<td>0.41</td>
<td>0</td>
<td>1.43</td>
<td>0</td>
<td>0.45</td>
</tr>
<tr>
<td>applu</td>
<td>1.13</td>
<td>0</td>
<td>0.32</td>
<td>0</td>
<td>0.86</td>
<td>0</td>
<td>0.31</td>
<td>0</td>
<td>1.79</td>
<td>10</td>
<td>0.33</td>
</tr>
<tr>
<td>mgrid</td>
<td>2.79</td>
<td>10</td>
<td>0.28</td>
<td>0</td>
<td>3.35</td>
<td>20</td>
<td>0.28</td>
<td>0</td>
<td>6.00</td>
<td>40</td>
<td>0.30</td>
</tr>
<tr>
<td>top 5 average</td>
<td>1.86</td>
<td>8</td>
<td>2.61</td>
<td>16</td>
<td>3.86</td>
<td>30</td>
<td>5.59</td>
<td>42</td>
<td>6.11</td>
<td>52</td>
<td>9.09</td>
</tr>
<tr>
<td>average</td>
<td>1.86</td>
<td>4</td>
<td>1.57</td>
<td>8</td>
<td>2.94</td>
<td>20</td>
<td>3.07</td>
<td>21</td>
<td>4.78</td>
<td>36</td>
<td>4.89</td>
</tr>
</tbody>
</table>
TABLE IV  
MPA AND SPI PREDICTION WHEN PROCESSES RUN WITH ART

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Extra</th>
<th>Extra</th>
<th>CAMP</th>
<th>AB</th>
<th>MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>17.40</td>
<td>72.01</td>
<td>-1.96</td>
<td>+4.89</td>
<td></td>
</tr>
<tr>
<td>mcf</td>
<td>16.72</td>
<td>72.62</td>
<td>-1.52</td>
<td>+2.38</td>
<td>-7.16</td>
</tr>
<tr>
<td>bzip2</td>
<td>6.13</td>
<td>31.48</td>
<td>+0.52</td>
<td>-0.13</td>
<td>-2.20</td>
</tr>
<tr>
<td>equake</td>
<td>10.92</td>
<td>48.03</td>
<td>+0.60</td>
<td>+0.19</td>
<td>-8.03</td>
</tr>
<tr>
<td>mesa</td>
<td>2.33</td>
<td>13.93</td>
<td>-0.33</td>
<td>+5.60</td>
<td>-2.56</td>
</tr>
<tr>
<td>vpr</td>
<td>8.41</td>
<td>42.24</td>
<td>+0.03</td>
<td>-0.66</td>
<td>-0.07</td>
</tr>
<tr>
<td>ammp</td>
<td>5.42</td>
<td>32.84</td>
<td>-2.33</td>
<td>+4.45</td>
<td>-5.54</td>
</tr>
<tr>
<td>mgrid</td>
<td>7.76</td>
<td>37.85</td>
<td>+2.17</td>
<td>-5.01</td>
<td>-3.29</td>
</tr>
<tr>
<td>applu</td>
<td>9.40</td>
<td>44.74</td>
<td>+2.48</td>
<td>-6.38</td>
<td>-5.83</td>
</tr>
</tbody>
</table>

Figure 4. Performance degradation for <art, mcf> pair.  
Figure 5. Performance degradation for <art, vpr> pair.  
Figure 6. Performance degradation for <vpr, mcf> pair.  

Figure 7. Profiled cache miss rate corresponding to effective cache size for different cache configurations.

4 MB and 6 MB of L2 unified cache. The three cache miss rate curves closely match each other, suggesting that process characterization data derived on one machine might be used to accurately predict the performance of cache-sharing processes on different types of processors with different cache structures.

VII. CONCLUSION  
Cache contention among processes running on different CMP cores heavily influences performance. A cache-contention aware assignment algorithm can help improve system throughput and reduce power consumption. However, this requires a model of cache contention behavior that can quickly and accurately determine the impact of different assignments on performance. This is challenging due to the large numbers of potential assignments of processes to CMPs. We have described CAMP, a predictive model that allows fast and accurate estimation of system performance degradation due to cache contention. More specifically, it first determines a process-dependent feature vector and reuse distance histogram via (potentially on-line) pre-characterization. The feature vectors of cache-sharing processes are supplied into a group of non-linear equations that determine the steady-state effective cache size and performance of each process. We also described a method to automate the profiling and performance prediction process. We evaluated the proposed technique on 55 different combinations of 10 SPEC CPU2000 benchmarks on a dual-core machine. The average performance prediction error is 1.57%. We also tested the generality of the proposed technique by profiling processes on one CMP and using the profiling information for performance prediction on two other CMPs with different cache sizes. In contrast with existing work, the proposed approach requires access only to information...
Conclusion

Summary

- Predictive model of contention on shared last-level cache
- Automated profiling and extraction of feature vector
- No modification of hardware or operating system
- “Average” error of <1.6%
Conclusion

Summary
- Predictive model of contention on shared last-level cache
- Automated profiling and extraction of feature vector
- No modification of hardware or operating system
- “Average” error of <1.6%

Discussion
- Varying input data
- Benchmarking crimes
- Generalisation
- Practical application